What is claimed is:

1. A method comprising:

extracting an asynchronous signal from a memory access instruction in a program to represent a latency of the memory access instruction; and

generating a wait instruction to wait for the asynchronous signal.

2. The method of claim 1, further comprising:

enforcing a first dependence between the memory access instruction and the wait instruction via the asynchronous signal.

3. The method of claim 1, further comprising:

introducing a pseudo signal to enforce a second dependence between the wait instruction and a memory access dependent instruction.

4. The method of claim 1, further comprising:

making the memory access instruction define the asynchronous signal; and

making the wait instruction use the asynchronous signal.

5. The method of claim 1, further comprising:

making the wait instruction define a pseudo signal; and

making an instruction that depends on the completion of the memory access instruction use the pseudo signal.

6. The method of claim 1, further comprising:

storing the asynchronous signal in a signal register of a network device.

7. The method of claim 3, further comprising:

storing the pseudo signal in a pseudo signal register of a network device.

8. A method, comprising subject to a dependence constraint of a program:

performing a first code motion on a first set of one or more instructions except each memory access instruction in the program, and

performing a second code motion on a second set of one or more instructions except each wait instruction in the program, to increase a number of instructions between issue and completion of the memory access instruction.

9. The method of claim 8, wherein the first code motion comprises moving the first instruction set forward through one or more paths of the program with the memory access instructions fixed, and the second code motion comprises moving the second instruction set backward through the one or more paths of the program with the wait instructions fixed.

10. The method of claim 8, wherein the first code motion comprises sinking the one or more instructions in the first set that occur in each predecessor block of a successor block into the successor block, and the second code motion comprises hoisting the one or more instructions in the second set.

11. The method of claim 8, comprising:

performing a speculative code motion on a wait instruction, in response to determining that the wait instruction is absent in at least one predecessor blocks of a successor block.

12. The method of claim 8, comprising:

in response to determining that the number of occurrence of a wait instruction in predecessor blocks of a successor block is less than the number of the predecessor blocks, appending a compensation code for the wait instruction to one or more predecessors that lack the wait instruction;

removing the wait instruction from the predecessors; and

prepending an instruction instance of the wait instruction to the successor block.

13. A compiler, comprising:

a code motion unit to perform code motion in a program subject to a dependence constraint of the program to hide a latency of a memory access instruction in the program.

14. The compiler of claim 13, further comprising:

an intermediate language unit to represent a memory access instruction in a program with an asynchronous signal associated with a latency of the memory access instruction.

15. The compiler of claim 13, further comprising:

an intermediate language unit to define an asynchronous signal in the memory access instruction to represent the latency and to generate a wait instruction that uses the asynchronous signal.

16. The compiler of claim 13, further comprising:

an intermediate language unit to define a pseudo signal in a wait instruction associated with the memory access instruction and to make an instruction that depends on the memory access instruction use the pseudo signal.

17. The compiler of claim 13, wherein the code motion unit further to

move a wait instruction associated with the memory access instruction and a first set of one or more instructions in a first direction subject to the dependent constraint, with the memory access instruction fixed; and

move the memory access instruction and a second set of one or more instructions in the program subject to the dependent constraint in a second direction that is opposite to the first direction, with the wait instruction fixed.

18. The compiler of claim 13, wherein the code motion unit further to

sink a wait instruction associated with the memory access instruction and a first set of one or more instructions of the program from each predecessor block to a successor block at a merging point of the predecessor blocks subject to the dependence constraint of the program, in response to determining that each predecessor block comprises the wait instruction and the one or more instructions, with the memory access instruction fixed; and

hoist the memory access instruction and a second set of one or more instructions in the program subject to the dependent constraint, with the wait instruction fixed.

19. The compiler of claim 13, wherein the code motion unit further to perform a speculative code motion on a wait instruction associated with the memory access instruction, in response to determining that the wait instruction is present in a first predecessor block of a merging successor block of the program and is absent in a second predecessor block of the merging successor block.

20. The compiler of claim 13, wherein the code motion unit further to recognize a wait instruction associated with the memory access instruction as a motion candidate subject to a dependence constraint of the program;

in response to determining that the wait instruction is present in a first predecessor block of the merging successor block and is absent in a second predecessor block of the merging successor block, insert a compensation code for the wait instruction into the second predecessor block; and

sink the wait instruction into a merging successor block of the first and second predecessor blocks subject to the dependence constraint.

21. The compiler of claim 20, wherein the code motion unit further to hoist the memory access instruction subject to the dependence constraint.

22. A machine readable medium comprising a plurality of instructions that in response to being executed result in a computing device

determining a motion candidate from one or more predecessor blocks of a first block of a program based on a dependence constraint of the program; and

performing a code motion on an instruction corresponding to the motion candidate to hide a latency associated with a memory access instruction.

23. The machine readable medium of claim 22, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

in response to determining that a number of occurrence of the candidate in the predecessor blocks is smaller than a number of predecessor blocks and in response to determining that the candidate is a wait instruction, appending a compensation code to one or more of the predecessor blocks where the candidate is absent.

24. The machine readable medium of claim 23, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

appending a wait instruction corresponding to the candidate to each of said one or more predecessor blocks where the candidate is absent.

25. The machine readable medium of claim 24, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

sinking each wait instruction corresponding to the candidate in each predecessor blocks of the first block into the first block.

26. The machine readable medium of claim 22, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

in response to determining that a number of occurrence of the candidate in the predecessor blocks equals to a number of the predecessor blocks, removing each instruction corresponding to the candidate from each predecessor block of the first block; and

prepending an instruction instance of the candidate to the first block.

27. The machine readable medium of claim 26, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

updating a dependent constraint of predecessor blocks of the first block.

28. The machine readable medium of claim 22, wherein the machine readable medium further comprising instructions that in response to being executed result in the computing device

determining a sinking candidate from one or more instructions of the program except the memory access instruction, based on a dependence constraint of the program;

performing a code sinking on each instruction corresponding to the sinking candidate subject to the dependence constraint;

determining a hoisting candidate from one or more instructions of the program except a wait instruction associated with the memory access instruction, based on the dependence constraint of the program; and

performing a code hoisting on each instruction corresponding to the hoisting candidate subject to the dependence constraint.